REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE Nov 98	3. REPORT TYPE AND DATES COVERED SBIR Phase I Final Report, Apr 98 – Oct 98

4. TITLE AND SUBTITLE
Final Report, Development of Combined Performance and Dependability Analysis Tools for Flight Critical Avionic Systems

5. FUNDING NUMBERS
C: N68335-98-C-0207

6. AUTHOR(S)
Dong Tang, Herbert Hecht

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
SoHaR Incorporated
8421 Wilshire Boulevard, Suite 201
Beverly Hills, CA 90211

8. PERFORMING ORGANIZATION REPORT NUMBER
J1096

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Naval Air Warfare Center – Aircraft Division
Building 2187 – Suite 3240D, 48110 Shaw Road, Unit 5
Patuxent River, MD 20670-1906
Attn: Sigmund G. Rafalik

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

19981201 009

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Report developed under SBIR contract. The object of this SBIR Phase I research is to develop fault tolerance analysis tools for flight critical avionic systems. The report demonstrates the feasibility of integrated performance and dependability modeling, a need that has not been adequately met. Evaluation metrics were developed and representative performance/dependability modeling tools were evaluated through case studies. The evaluation identified two approaches that can overcome the "time explosion" problem encountered in modeling performance (high occurrence rate) and dependability (very low occurrence rate) events: (1) combining discrete event performance simulation and analytical dependability modeling, and (2) fault injection simulation with importance sampling. The report shows how to relate performance measures (system capacity) to system states in terms of component failures in implementing the first approach. The "performance loss" measure is introduced and the likelihood ratio for this measure is generated to support the second approach. To automate these approaches, a tool set that integrates a commercial performance simulation tool and a commercial dependability modeling tool is proposed. Necessary improvements/extensions to these tools are identified. The proposed tool set is suitable for military avionic systems as well as commercial fault tolerant systems.

14. SUBJECT TERMS
SBIR Report, Avionic System, Performance, Dependability, Modeling, Evaluation, Simulation, Software Tool, Fault Tolerance

15. NUMBER OF PAGES 33

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited

Standard form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

REF D

Development of Combined Performance and Dependability Analysis Tools for Flight Critical Avionic Systems

SBIR Phase I Final Report
Contract N68335-98-C-0207[1]

Prepared for

Sigmund Rafalik

Naval Air Warfare Center
Aircraft Div., Systems Engineering Dept.
System Development and Integration Branch
Patuxent River, Maryland

by

Dong Tang
Herbert Hecht

SoHaR Incorporated
Beverly Hills, California

November 1998

[1]This work was supported by the U.S. Naval Air Warfare Center under Contract N68335-98-C-0207. However, the opinions and viewpoints expressed herein are the authors' personal ones and do not necessarily reflect those of the U.S. Navy.

Abstract

This SBIR Phase I research demonstrates the feasibility of integrated performance and dependability modeling, a need that has not been adequately met. The integrated modeling supports decision makers who must evaluate, e. g., the capability of dispatching *x* aircraft per hour. Evaluation metrics were developed and representative performance/dependability modeling tools were evaluated through case studies. The evaluation identified two approaches that can overcome the "time explosion" problem encountered in modeling performance (high occurrence rate) and dependability (very low occurrence rate) events: (1) combining discrete event performance simulation and analytical dependability modeling, and (2) fault injection simulation with importance sampling. The report shows how to relate performance measures (system capacity) to system states in terms of component failures in implementing the first approach. The "performance loss" measure is introduced and the likelihood ratio for this measure is generated to support the second approach. To automate these approaches, a tool set that integrates a commercial performance simulation tool and a commercial dependability modeling tool is proposed. Necessary improvements/extensions to these tools are identified. The proposed tool set is suitable for military avionic systems as well as commercial fault tolerant systems. The research advances the state-of-the-art in modeling fault tolerant systems, thereby improving fault tolerance effectiveness and lowering development cost.

Table of Contents

1. Introduction

This SBIR Phase I research has shown the feasibility of combining performance and dependability modeling in coordinated software tools, a capability that is needed but has not been adequately met. The need for modeling in the broadest sense arises from the complexity of computer based systems in which neither functional nor quality-of-service attributes can be evaluated by conventional analysis. Naval aviation uses some of the most advanced computer based systems and the need for modeling tools to support design trade-offs and system evaluation is therefore particularly urgent.

The combined performance and dependability modeling addresses the need of decision makers who evaluate the ability of a system to handle a threat of *x* incoming missiles, or dispatching *y* aircraft per hour, or of handling *z* messages per minute. In all of these cases a component or software failure as well as impaired performance can jeopardize the ability of a system to meet mission objectives. Actions taken to improve dependability can detract from performance (e. g., the message exchange in voting structures), and measures taken to improve performance can detract from dependability (e. g., speeding up transmission rates will increase errors due to noise). These interactions make a combined model very valuable not only in naval aviation but also in a wide array of other military and commercial applications.

The technical challenge in developing an integrated performance and dependability model arises from the difference in the time scales of the two attributes. Performance modeling is typically concerned with events that occur millions of times per second whereas dependability modeling deals with event (failure) rates of one per 10^3 to 10^6 hours. A model that provides high fidelity for performance will have to perform astronomical numbers of iterations to generate events that affect dependability. This "time explosion" problem has been partially addressed in the discrete event simulation [Goswami97] and in the large-scale Markov modeling [Conway87] by using the importance sampling method or other techniques but neither of these references provides a comprehensive approach for combined performance and reliability modeling.

The results of our research apply to two major modeling techniques: analytical models and discrete event simulations. The former use mathematical relationships, typically the probability of transitions from one state to another, to compute the overall probability of arriving at a specified outcome. When applied to dependability modeling the inputs are the probability of failure of individual parts or assemblies, and the result is the probability of being in a specified degraded or failed state. Applied to performance modeling, the inputs are the probability of specific operational demands (message lengths, addresses, etc.) and the output is the probability of handling the required number of operations. Because the analytical models operate on specified probabilities, the time explosion is primarily a computational accuracy problem that can be resolved by established techniques. Discrete Event Simulations (DES) work not only with transition probabilities but also with time delays between events, and it is for the latter that time explosion creates a major problem.

In the past, dependability evaluation has primarily utilized analytical models while performance evaluation has utilized discrete event simulations. During the early life cycle phases (feasibility studies through preliminary design) analytical models are adequate for most combined modeling needs but because the performance modeling community is used to discrete event simulations we are proposing combining the performance DES with an analytical dependability model. For later life cycle phases we are proposing importance sampling (explained later) to reduce the time explosion problem and thus to facilitate the development of an integrated discrete event simulation for performance and dependability. Use of DES for dependability modeling permits injection of selected faults at specific steps in the processing cycle, and this an important advantage over analytical models during the later development phases (e. g., for trouble shooting).

Thus, this report demonstrates two approaches for integrated performance/dependability evaluation:

- Combined use of performance simulation and analytical dependability modeling.
- Applying the importance sampling method to the fault injection simulation to increase the probability of failures and thereby reduce the discrepancy in occurrence rates between dependability and performance events.

For the first approach, we show how to relate performance measures (system capacity) to system states that are defined in terms of component failures. For the second approach, we introduce the performance loss measure and identify the likelihood ratio function for this measure, which can be used to calculate the real probability of outcomes based on results obtained from a simulation without fault injections and from a simulation with fault injections at an accelerated fault rate. We also identify a tool set which includes a commercial performance simulation tool and a commercial dependability modeling tool to implement the above approaches in the Phase II effort.

The results of this research also advance the state-of-the-art in software tools for modeling fault tolerant (FT) systems, thereby promoting improved fault tolerance effectiveness and lower life cycle cost. Established FT techniques can increase reliability, but the increased acquisition cost, weight, power consumption and maintenance requirements may be unacceptable in high performance airframes. Moreover, the increasing emphasis on COTS components means that the designer must now utilize (and sometimes work around) FT capabilities provided by these components rather than being completely free to optimize the fault tolerance for the system. To compare alternative designs based on COTS components models are needed to select the most cost-effective configuration that meets all performance and dependability requirements. Thus, a new generation of modeling tools based on our research will provide much needed capability for high fidelity modeling of fault tolerant systems.

In this report the term *dependability*[2] designates the attributes that are to be achieved by the fault tolerance provisions. In a fault tolerant system, system performance may degrade because of faults and handling of faults by fault tolerance mechanisms. Thus, traditional performance evaluation should be augmented to include dependability evaluation and the tool set should facilitate this modeling approach. In fault tolerant computing research, performance measures evaluated considering fault conditions are categorically called *performability* [Meyer80], and performability modeling typically uses an analytical approach (mathematical models and solution methods).

[2]The *dependability* concept was proposed in the *15th International Symposium on Fault-Tolerant Computing* (FTCS-15) [Laprie85] and revised in FTCS-25 [Laprie95]. Dependability is defined as the "property of a computer system such that reliance can justifiably be placed on the service it delivers." Major measures of dependability include *availability*, *reliability*, *safety*, and *maintainability*.

Analytical dependability modeling tools are primarily aimed at *structural* FT provisions, such as active/standby (A/S), triple modular redundancy (TMR), or dual-dual (D/D) redundancy, all of which involve addition of physical components. The multiplicity of computing and communication components in current aircraft makes it possible to re-assign computational tasks in case of a computer outage, or to re-route communications when a link is down without adding physical resources. Simulation tools should have capabilities to model not only structural redundancy, but also rollback and retry (an effective FT technique for many software and some transient hardware failures), shifting processing from a failed computer to a functioning one, or re-routing around a link that has experienced an outage. Such *functional* FT can be expected to play an increasingly important part in future military as well as commercial aircraft.

In the rest of this report, Section 2 generates criteria for evaluating candidate modeling tools. Section 3 uses the generated criteria and case studies to evaluate available tools to form a basis for modifying them and synthesizing a more suitable tool set. Section 4 describes a high level design of a tool set which integrates a commercial performance modeling tool with a commercial dependability modeling tool. Section 5 provides concluding remarks.

2. Evaluation Metrics

First, we develop metrics for evaluating performance/dependability modeling tools. The metrics are classified into two groups: the benefit metrics and the cost metrics, as shown in Figure 1. The benefit group includes four categories: functionality, user interface, validation means, and integration. The cost group includes two categories: acquisition cost and operating cost (primarily represented by the required execution time (performance). Details of these categories are discussed below.

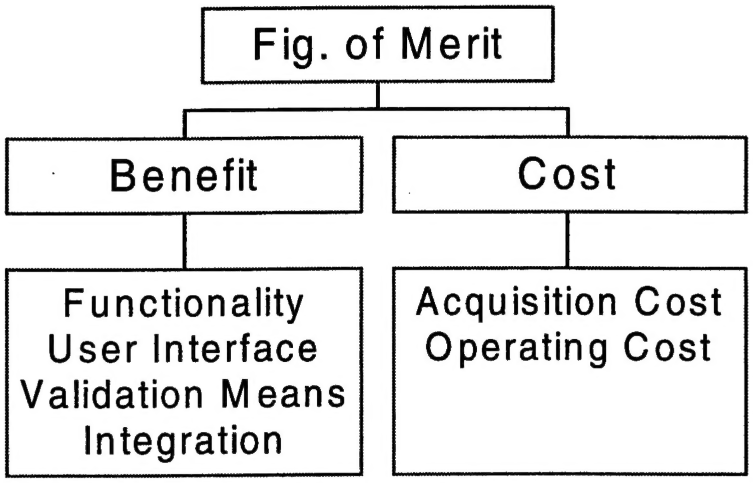

Figure 1 Metrics Categories

2.1 Functionality

The functionality category consists of the following metrics

- Model types
- Hierarchical modeling approach
- Measures generated
- Statistical functions supported
- Parametric analysis capability
- Library of primitive fault tolerance models

Commonly used model types are: combinatorial models (e.g., reliability block diagrams and fault trees), state space based models (e.g., Markov chains and Petri nets), and discrete event simulations (DES). If failures of components in the modeled system are independent, the simpler structures of combinatorial models suffice. This is usually the case for the high-level system modeling in which subsystem failures can be considered independent. If the failure of a component may affect other components, or a reconfiguration is involved upon a component failure, state space models are required. This is usually the case for the subsystem or lower level modeling where failure interactions among components are strong. Both combinatorial and state space models are analytical models that use (failure or event) probabilities. DES not only permits probabilistic modeling, it also permits behavioral modeling which does not require that the effect of faults be predefined. DES is suitable for performance modeling and fault injection studies at detailed levels. These model types are complementary to each other and the combined use of these models can often be the most cost-effective approach to evaluating performance oriented dependability (performability).

Because a computing system is organized hierarchically, the model to characterize the system should be organized in the same manner. The hierarchical modeling approach is recommended because it reduces model complexity and it facilitates identifying problem areas (by comparing results from different submodels). Different submodels may use different model structures, depending on the architecture and behavior of the modeled subsystems.
For repairable systems, measures generated should include availability and Mean Time Between Failures (MTBF). For non-repairable systems, measures generated should include reliability and Mean Time To Failure (MTTF). When performance is incorporated in the evaluation, the tool should be able to generate both steady state and transient performability measures. The output precision should be high enough for evaluating high availability/reliability systems. Generally, a range that permits good representation of failure probabilities of 10^{-8} is desired.

The Statistical functions supported should include the exponential, Weibull, normal, and lognormal, because these are the most representative distributions in characterizing failure inter-arrival times and repair times [Iyer96].

Parametric analysis means that the modeling tool can generate multiple sets of outputs for varying parameters. The varied parameter can be a failure/repair rate, a fault tolerance coverage, a redundancy parameter (k-out-of-n), or time.

A library of primitive dependability models for typical fault-tolerance architectures, such as primary/standby modular redundancy and triple modular redundancy models, provides a collection of basic blocks from which users can develop their own models. These reusable components reduce the model development time. The library should be easily expandable.

2.2 User Interface

The user interface category consists of the following metrics:

- Graphical input of models
- Graphical output of results
- Model editing functions
- On-line help information
- Quality of user's manual

Graphical input of models helps to make them easily understandable. It also reduces errors in model construction (compared with the text input method). The graphical models should be exportable for documentation. Graphical output of results is used for displaying multiple sets of results such as those generated by parametric analysis.

Model editing functions include adding, deleting, revising, moving, and copying objects. The on-line help information should include definitions for all commands and descriptions for model structures supported by the tool. A user friendly help system should also provide index and search capabilities.

The user's manual should be well organized and incorporate illustrative examples.

2.3 Validation Means

The validation means include:

- Built-in consistency checking
- Ability to output results from each hierarchical level

The built-in consistency checking normally checks correctness for model structures. If possible, it should also check correctness of parameters, e.g., by reasonableness tests. Ability to output results for all levels can serve at least two purposes: (1) verify models and parameters from the bottom to the top level, and (2) identify the weakness of the modeled system.

2.4 Integration

Integration means:

- Compatibility with documentation tools
- Integratability with user's existing performance modeling tools

The performance/dependability modeling tools should be easily integrated with widely used documentation tools (e.g., MS Word). The "copy and paste" approach is preferred for transferring model diagrams and results from a modeling tool to a documentation tool. For example, the user clicks the "Copy to Clipboard" command in the modeling tool to copy a model diagram or a set of results to the clipboard and then clicks the "Paste" command in the documentation tool to paste it into the document being edited by the documentation tool.

In the current market, commercial performance modeling tools are more mature than commercial dependability tools. Users of the commercial performance modeling tools have developed a large amount of performance models and many problems still exist in incorporating dependability modeling tools in these models. The integration of candidate dependability modeling tools with the existing commercial performance modeling tools is a preferred approach for these users. One way to integrate is to have a pipe which transfers the results generated by the performance modeling tools into the dependability modeling tool as parameters in the dependability model. Another way to integrate is to include dependability evaluation techniques (e.g., fault injection) in the performance modeling tools to make them support performability modeling.

2.5 Acquisition Cost

The acquisition cost includes:

- Cost of the tool
- Cost of the platform

The cost of the tool and the platform to support the tool is an important factor for commercialization.

2.6 Operating Cost

The operating cost measures include:

- Execution time (performance of the tool)
- Reliability

The execution time for most applications should be of a reasonable duration. The execution time for a typical application can be compared among similar tools. The candidate tool should be reasonably reliable, i.e., applications should be developed and execute without experiencing crashes.

3. Evaluation of Selected Tools

The selected tools for evaluation in this project include: Foresight [NuThena96], Designer [Cadence98], MEADEP[Tang98, Tang99], and SHARPE [Sahner87, Sahner96]. These tools are representative of the three types of models: combinatorial models, state space based models, and simulation models. Foresight and Designer are commercial performance simulation tools that are currently used in airspace and aerospace applications. MEADEP and SHARPE are dependability modeling tools that support both combinatorial and state space based models.

The inclusion of Foresight and Designer made the other two simulation tools, DEPEND [Goswami97] and UltraSAN [Sanders95], the candidate tools in the Phase I proposal, not possible for inclusion in the evaluation because of the limited scope of this project. The consulting services associated with these tools were thus no longer necessary. There are also other technical reasons for not including these tools in the evaluation. DEPEND is a functional fault injection simulation tool which could be expanded to include performance modeling. Since both Foresight and Designer have been used by the Navy and a large amount of performance models have been developed using these tools, incorporating fault injection techniques in these tools is a more favorable approach than incorporating performance evaluation in other fault injection simulation tools. UltraSAN is a Stochastic Activity Networks (a stochastic variant of Petri nets) based simulation tool. Although the Petri net is a powerful model type, it is not included in this evaluation for the following reasons:

- The Petri net is more a specification notation than a mathematical model. A Petri net model cannot be solved mathematically. The model needs to be either solved by simulation or translated into a Markov model which is then solved by numerical methods. When the model is large, the only feasible way to solve it is through simulation. In this case, a Petri net model becomes a special simulation model. Compared with general simulation models supported by discrete event simulation tools such as Foresight and Designer, Petri net models are much less flexible and more difficult to understand (e.g., places/transitions/tokens vs. various objects such CPU, memory, and link). That is, the capabilities of Petri nets are a subset of those of discrete event simulation models while Petri nets do not generally have advantages in model solution because large models have to be solved by the same simulation method used for general discrete simulation models. For small models, Markov chains can be directly specified without having to use Petri nets.

- Our experience showed that sometimes it is difficult to use the Petri net notation to specify a system state based performability model. For instance, in the case study 2 to be discussed later, a Markov reward model is used to evaluate performability. Each state in the Markov model represents a system state for which the performance has been evaluated by a DES. The reward associated with that state represents the system performance level (processing capacity). Specifically, state i means that there are i processors functioning properly in the system. The probability of being in state i multiplied by the reward (performance) is the *performability* associated with state i. The summation of the performability measures over all states is the performability of the modeled configuration. When we tried to translate this Markov reward model to a Generalized Stochastic Petri Net model (GSPN, a super set of Petri net models) supported by SHARPE, a single "place" representing system failure states was generated (a conventional way to model system failures with Petri nets). If i tokens arrive in the place, it means that *i* processors have failed. Since the place can represent several different system failure states (one processor failure, two processor failures, etc.), it is difficult to assign a reward rate or performance level to the place because each performance level is unique to a single system state. Therefore at least this GSPN model is not well suited for use in this proposed performance simulation/analytical dependability modeling approach.

- The Phase II effort of this research is intended to develop an automation engine for generating models based on a series of queries and answers collected through a user friendly interface. The generated dependability models will be solved numerically to obtain results. If the generated models are in the Petri net format, the models have to be translated to the Markov chain format before they are solved numerically. Since Markov reward models can be directly generated by the automation engine, there is no need to first generate a Petri net and then to translate it to a Markov chain.

3.1 Functional Evaluation

Table 1 is a summary of features of the selected tools in terms of the evaluation metrics developed in the previous section.

Table 1 Summary of Features for Selected Tools

Metric \ Tool	**Foresight**	**Designer**	**MEADEP**	**SHARPE**
Functionality	Simulation with embedded programming; Expon, Gaussion; No primitive FT models	Simulation without embedded programming; Expon, Gaussion; No primitive FT models	RBD, Markov; Expon, Weibull; Precision: 10 digits; Some primitive FT models	RBD, Markov, Petri net, fault tree, etc.; Expon polynomials; Precision: 8 digits; No primitive FT models
User Interface	Graphical input/output; On-line help without index and search	Graphical input/output; On-line help without index and search	Graphical input/output; On-line help with partial support for search	No graphical input/output; No on-line help
Validation Means	Model consistency checks;	Model consistency checks	Model consistency checks; Partial parameter consistency checks	Model consistency checks at runtime
Integration	Postscript export; Used by customers	Postscript export; Used by customers	WMF export	No export
Acquisition Cost	High ($35,000); UNIX, NT/Exceed	High ($35,000); UNIX	Low ($2,000); Windows 95/98/NT	Medium ($10,000); UNIX
Performance	Slow	Fast	Fast	Fast

The two simulation tools in the table, Foresight and Designer, are similar in many aspects. However, because Foresight supports customized Ada code embedded in a user defined simulation block (which we call the *embedded programming*), it should have more powerful modeling capabilities than Designer. But the execution speed of Foresight is much slower than that of Designer. The developer of Foresight claims that CoderC, an optional supporting component of Foresight (discussed in Section 4) provides significantly faster execution but we were unable to verify that.

Comparing the two analytical dependability modeling tools, we see that SHARPE supports more model types than MEADEP, but it lacks a user friendly interface, validation means, and the integration capability with other tools. In our experience, reliability block diagrams and Markov models together, which are supported by MEADEP, provide adequate modeling capabilities for typical applications.

The above analysis in terms of our metrics provides a preliminary evaluation of the selected tools. In the following subsections, we perform an experimental evaluation through case studies to obtain more insights into these tools.

3.2 Case Study 1: Modeling of a Dual-Dual Redundant Flight Control System

In this case study, we develop a dependability model for the computer channels and their I/O interfaces in a flight control system shown in Figure 2 (represented by the central blocks). The

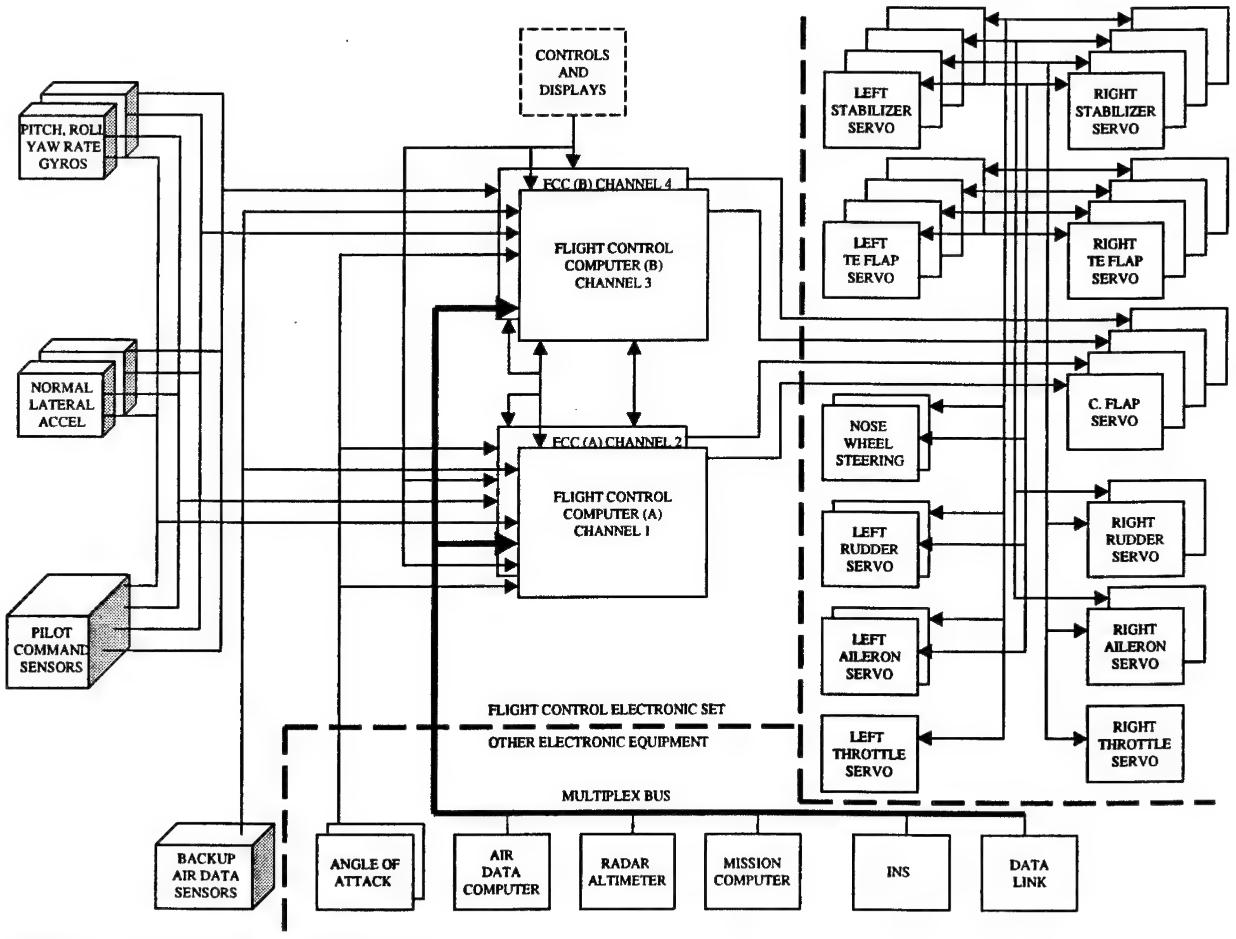

Figure 2 A Flight Control System

detailed architecture of the computer channels and their I/O interfaces are shown in Figure 3, which is a Dual-Dual Redundant Computer Subsystem (DDRCS). The model is implemented with MEADEP and SHARPE, and the two tools will be compared using this example. The MEADEP implementation of the model is shown in Figures 4-10.

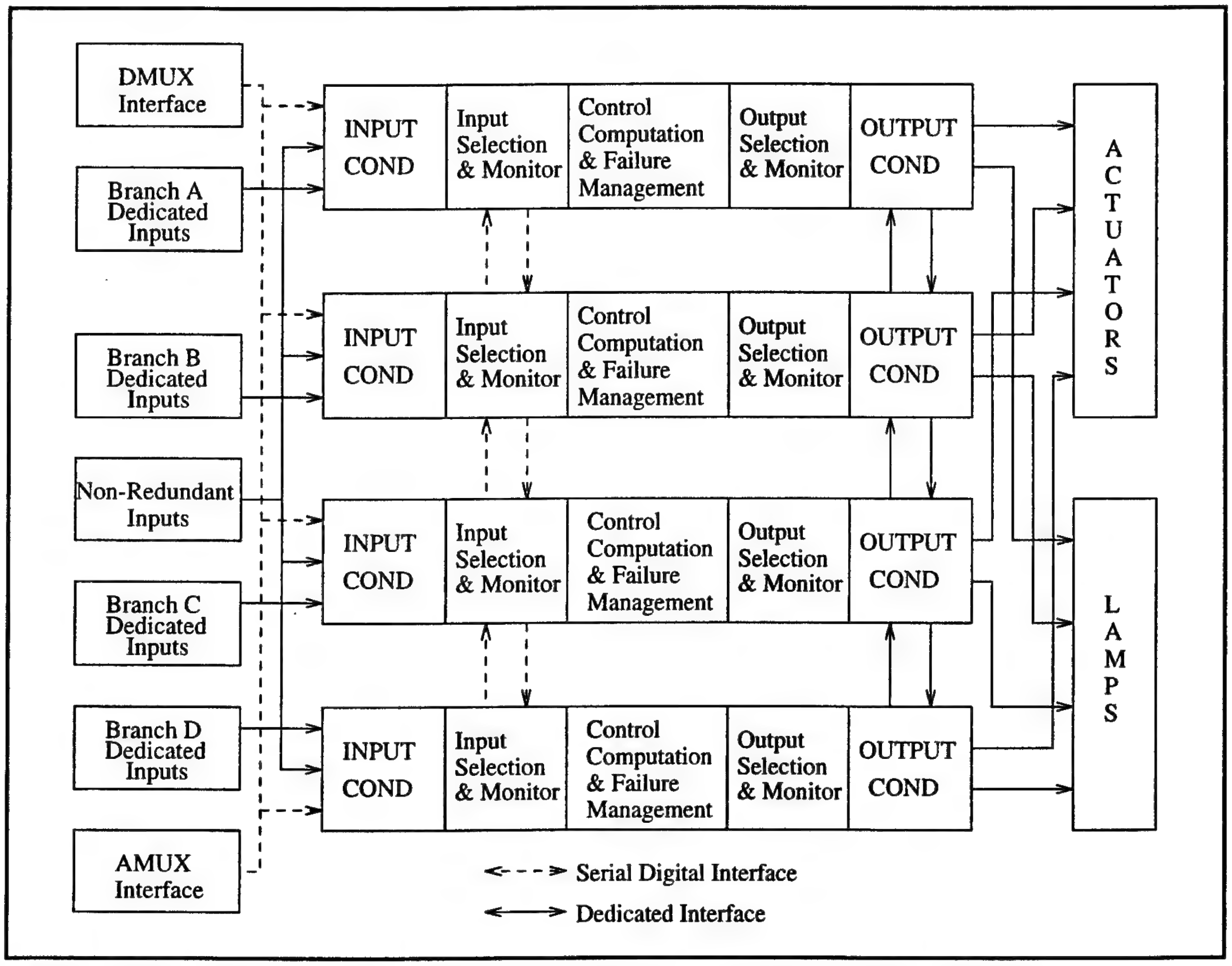
DMUX Interface
Branch A Dedicated Inputs
Branch B Dedicated Inputs
Non-Redundant Inputs
Branch C Dedicated Inputs
Branch D Dedicated Inputs
AMUX Interface
INPUT COND
Input Selection & Monitor
Control Computation & Failure Management
Output Selection & Monitor
OUTPUT COND
ACTUATORS
LAMPS
Serial Digital Interface
Dedicated Interface

Figure 3 Dual-Dual Redundant Computer Subsystem in the Flight Control System

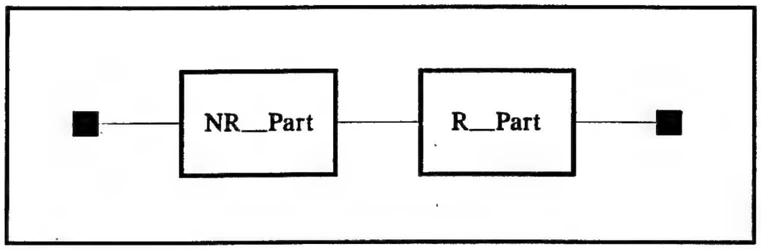

Figure 4 Top-Level System Model

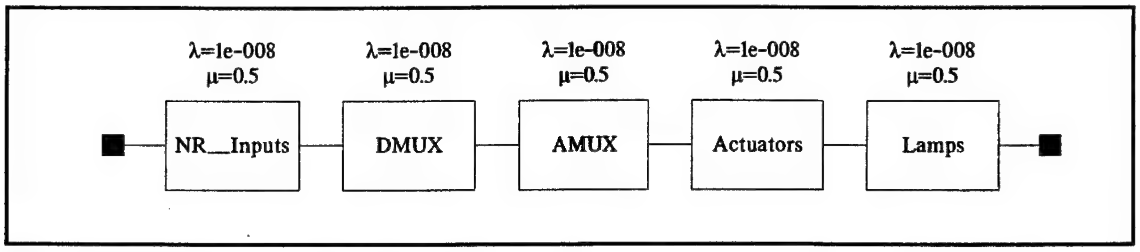

Figure 5 Non-Redundant Part Model

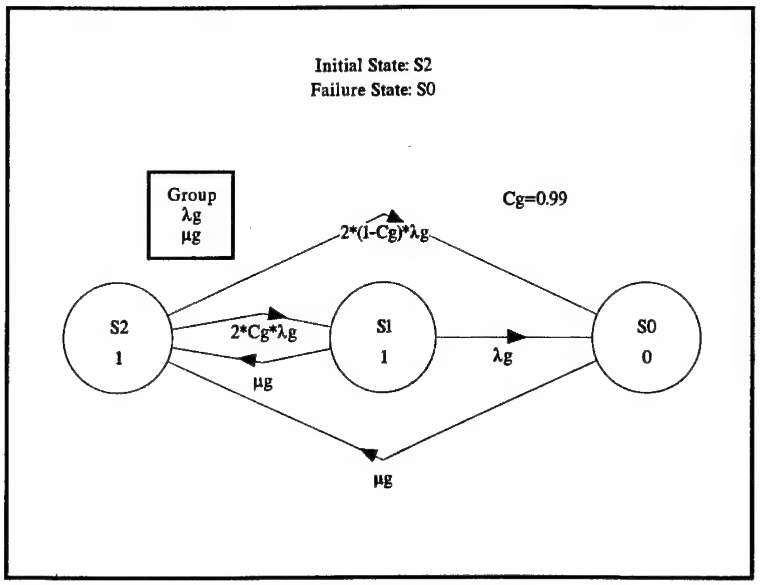

Figure 6 Redundant Part Model: 2 Groups

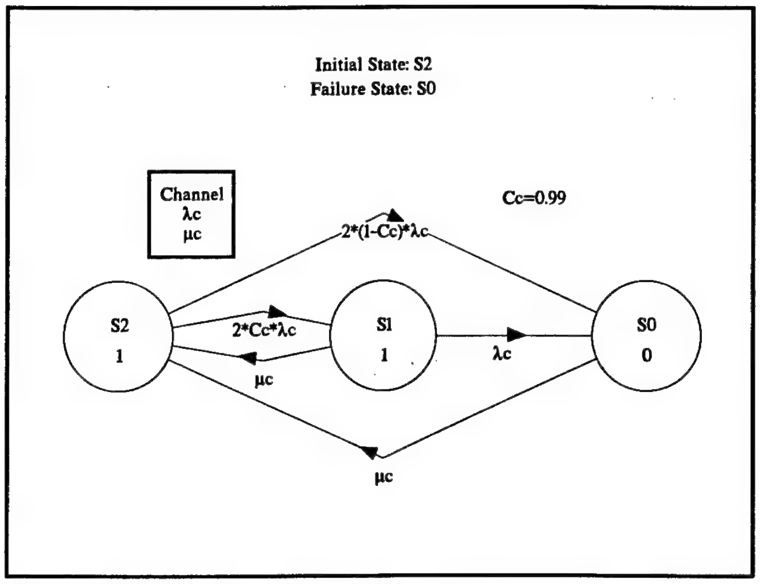

Figure 7 Group Model: 2 Channels

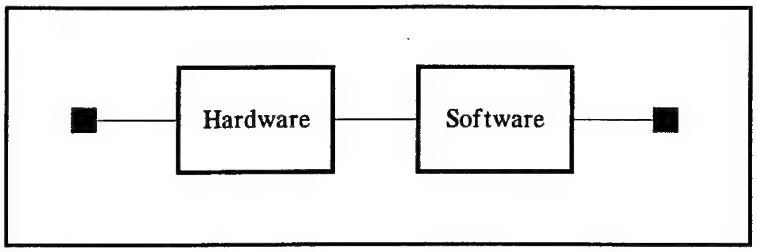

Figure 8 Channel Model

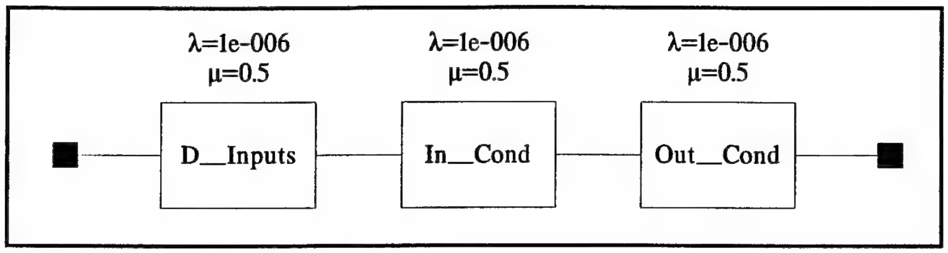

Figure 9 Channel Hardware Model

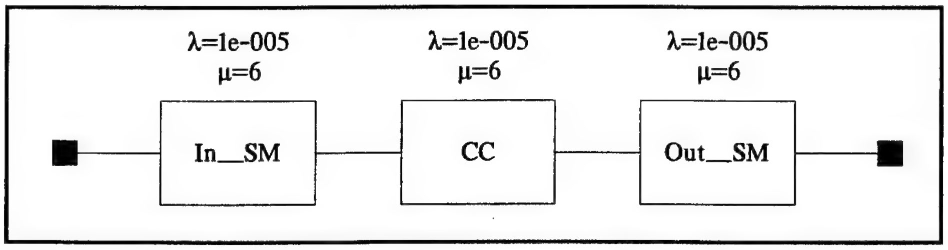

Figure 10 Channel Software Model

The top-level system model consists of two submodels: the redundant part and the non-redundant part (Figure 4). The non-redundant part consists of all non-redundant components in the system (Figure 5). The redundant part consists of two redundant groups (Figure 6). Each of the two groups consists of two redundant channels (Figure 7). Each channel includes three hardware components (Dedicated Inputs, Input Conditions, and Output Conditions) and three software components (Input Selection & Monitor, Control Computation & Failure Management, and Output Selection & Monitor) (Figures 8-10). In Figures 5, 9 and 10, a number like 1e-008 is generated by C++ and means 1.0×10^{-8}. The Markov models in Figures 7 and 8 are explained further as follows:

S_2	Normal state
S_1	State in which one channel (or group) has failed and the reconfiguration has been successful
S_0	State in which both channels (or groups) have failed
$\lambda_c(\lambda_g)$	Failure rate of a channel (or group)
$\mu_c(\mu_g)$	Recovery rate of a channel (or group)
$C_c(C_g)$	Coverage — probability that the reconfiguration is successful

A heavy frame means that the channel (group) failure rate $\lambda_c(\lambda_g)$ and recovery rate $\mu_c(\mu_g)$ are evaluated from the lower level model. The transition from S_2 to S_1 represents that one channel (group) has failed and the reconfiguration is successful. The transition from S_2 to S_0 represents that one channel (group) has failed and the reconfiguration is not successful, i.e., both channels (groups) have failed. In state S_1, the failed channel (group) is recovering at the rate of $\mu_c(\mu_g)$ which is modeled by the transition from S_1 to S_2. During this recovery process, the channel (group) that is working can also fail, which is modeled by the transition from S_1 to S_0. The transition from S_0 to S_2 represents that the system is restored to the normal state. The rate for this transition is $\mu_c\,(\mu_g)$ which assumes that there are two repair persons or that the repair is just a restart of the two channels (groups). If there is only one repair person, $2\times\mu_c\,(\mu_g)$ should be used.

From this graphical representation, MEADEP can generate a text modeling file for availability evaluation (for a repairable system) or a text modeling file for reliability evaluation (for a non-repairable system). The user does not have to change the model itself. However, when this model is implemented with SHARPE, the user has to define one model for availability evaluation and another model for reliability evaluation. The SHARPE implementation of the DDRCS availability model is shown in Table 2. The SHARPE implementation of the DDRCS reliability model is shown in Table 3.

Table 2 The SHARPE DDRCS Availability Model

```
* Markov Chain for R_Part Model
markov R_Part
S2  S1  2*Cg*Lg
S2  S0  2*(1-Cg)*Lg
S1  S2  Mg
S1  S0  Lg
S0  S2  Mg
reward
S2  1
S1  1
S0  0
end

* Markov Chain for Group Model
markov G_Model
S2 S1  2*Cc*Lc
S2 S0  2*(1-Cc)*Lc
S1 S2  Mc
S1 S0  Lc
S0 S2  Mc
reward
S2 1
S1 1
S0 0
end

* Channel Hardware Model
bind
Lhc 0.000001
Mhc 0.5
Ahc Mhc/(Lhc+Mhc)
Ahw Ahc^3
Uhw 1-Ahw
Mhw Mhc
Lhw Uhw*Mhw/Ahw
end

* Channel Software Model
bind
Lsc 0.00001
Msc 6
Asc Msc/(Lsc+Msc)
Asw Asc^3
Usw 1-Asw
Msw Msc
Lsw Usw*Msw/Asw
end

* Channel Model
bind
Ac Ahw*Asw
Uc 1-Ac
Mc  (Lhw*Mhw+Lsw*Msw)/(Lhw+Lsw)
Lc Uc*Mc/Ac
end
```

```
* Group Model
bind
Cc 0.99
Ag  exrss(G_Model)
Ug  1-Ag
Mg  Mc
Lg Ug*Mg/Ag
end

* R_Part Model
bind
Cg 0.99
Arp exrss(R_Part)
Urp 1-Arp
Mrp Mg
Lrp Urp*Mrp/Arp
end

* NR_Part Model
bind
Lnrc 0.00000001
Mnrc 0.5
Anrc Mnrc/(Lnrc+Mnrc)
Anrp Anrc^5
Unrp 1-Anrp
Mnrp Mnrc
Lnrp Unrp*Mnrp/Anrp
end

* Top Level Model
bind
Amain Arp*Anrp
Umain 1-Amain
Mmain  (Lrp*Mrp+Lnrp*Mnrp/(Lrp+Lnrp)
Lmain Umain*Mmain/Amain
end

* Output Results
echo
echo Hardware Availability/Unavailability
expr Ahw, Uhw
echo Software Availability/Unavailability
expr Asw, Usw
echo Channel Availability/Unavailability
expr Ac, Uc
echo Group Availability/Unavailability
expr Ag, Ug
echo R_Part Availability/Unavailability
expr Arp, Urp
echo NR_Part Availability/Unavailability
expr Anrp, Unrp
echo System Availability/Unavailability
expr Amain, Umain

end
```

It is seen that the SHARPE textual representation of the model is much less understandable than the MEADEP graphical representation. In order to define a model, the user has to learn the SHARPE language, which is much more difficult than using a "drag and drop" interface to build the model. In addition, the user must be an expert in reliability modeling. For example, the failure rate λ_g in Figure 5 is the failure rate of it sub-model — Group Model (Figure 6). In MEADEP, this is specified by the frame which includes the name Group and the parameter λ_g (Figure 5). But in SHARPE, the user has to know λ_g is the reciprocal of the mean time to the failure state (S_0) of the Group model and define an expression (1/mean(G_Model, S_0)) for it (Line 5 in the right column of Table 3).

Table 3 The SHARPE DDRCS Reliability Model

```
* Markov Chain for R_Part Model
markov R_Part
S2  S1  2*Cg*Lg
S2  S0  2*(1-Cg)*Lg
S1  S0  Lg
reward
S2  1
S1  1
S0  0
end
end

* Markov Chain for the Group Model
markov G_Model
S2  S1  2*Cc*Lc
S2  S0      2*(1-Cc)*Lc
S1  S0  Lc
reward
S2  1
S1  1
S0  0
end
end

* Hardware Model
bind
t   100
e   2.7182818
Lhc 0.000001
Rhc e^(-Lhc*t)
Rhw Rhc^3
Lhw 3*Lhc
end

* Software Model
bind
Lsc 0.00001
Rsc e^(-Lsc*t)
Rsw Rsc^3
Lsw 3*Lsc
end

* Channel Model
bind
Rc  Rhw*Rsw
Lc  Lhw+Lsw
end
```

```
* Group Model
bind
Cc  0.99
Rg  exrt(t, G_Model)
Lg  1/mean(G_Model, S0)
end

* R_Part Model
bind
Cg      0.99
Rrp exrt(t, R_Part)
Lrp 1/mean(R_Part, S0)
end

* NR_Part Model
bind
Lnrc    0.00000001
Rnrc    e^(-Lnrc*t)
Rnrp    Rnrc^5
Lnrp    5*Lnrc
end

* Top Level Model
bind
Rmain  Rrp*Rnrp
Lmain  Lrp+Lnrp
end

* Output Results
format 8
echo
echo Hardware Reliability & MTTF
expr   Rhw,    1/Lhw
echo Software Reliability & MTTF
expr   Rsw,    1/Lsw
echo Channel Reliability & MTTF
expr   Rc, 1/Lc
echo Group Reliability & MTTF
expr   Rg, 1/Lg
echo R_Part Reliability & MTTF
expr   Rrp,    1/Lrp
echo NR_Part Reliability & MTTF
expr   Rnrp,  1/Lnrp
echo System Reliability & MTTF
expr   Rmain, 1/Lmain

end
```

Table 4 compares some results generated by MEADEP and by SHARPE for the DDRCS model and shows that these match. Because the maximum number of digits after the decimal printed is 8 for SHARPE and 10 for MEADEP, MEADEP provides more accurate availability results (see availability values for the Group, R_Part and System models). The small differences (less than one hour) in the Mean Time To Failure (MTTF) are probably due to the different iteration algorithms used in the tools.

Table 4 Steady-State Results Generated by MEADEP and SHARPE

Tool	**MEADEP**		**SHARPE**	
Measure	**Availability**	**MTTF (No Repairs)**	**Availability**	**MTTF (No Repairs)**
Hardware Model	0.999994	333,333.3 hours	0.999994	333,333.3 hours
Software Model	0.999995	3,333.3 hours	0.999995	3,333.3 hours
Channel Model	0.999989	30,303.0 hours	0.999989	30,303.0 hours
Group Model	0.9999997798	45,151.7 hours	0.9999978	45,151.5 hours
R_Part Model	0.9999999956	67,276.3 hours	0.999999996	67,275.8 hours
NR_Part Model	0.9999999	20,000,000 hours	0.9999999	20,000,000 hours
System Model	0.9999998956	6,705.1 hours	0.999999896	6,705.0 hours
Execution Time	1 second	2 seconds	1 second	1 second
Platform	Pentium II 300, Windows NT		Sun SPARCstation 2, Solaris	

This example shows both tools have capabilities to model dependability for fault tolerant avionic systems with a hierarchy of combinatorial models and Markov models. The results match each other. However, the cost for developing the SHARPE model is higher than for the MEADEP model due to the following reasons:

- MEADEP provides a "drag and drop" interface for model input while SHARPE uses a complex modeling language.
- A single graphical representation in MEADEP can be used to generate both the executable availability model and the executable reliability model automatically while the two model types have to be defined separately in SHARPE.

Because of its graphical representation, the MEADEP model has a much better visibility than the SHARPE model. In addition, the SHARPE model developer needs to have a deep knowledge in dependability modeling to define a model, which is not required for the MEADEP model developer.

3.3 Case Study 2: Modeling of Processors in a UAV Control System

In this case study, we develop a performance-dependability model for the processors in a Unmanned Air Vehicle (UAV) control system. The performance evaluation uses simulation with Foresight and Designer while the dependability evaluation uses analytical modeling with MEADEP. The modeled system consists of three CPUs which process a number of signals for use in flight control and weapon control. The ability of the system to process these input requests is affected by the capacity of the processors. When failures occur on the processors, the capacity is degraded, and therefore the performance is degraded. This degraded performance, or performability, will be evaluated using the approach shown in Figure 11.

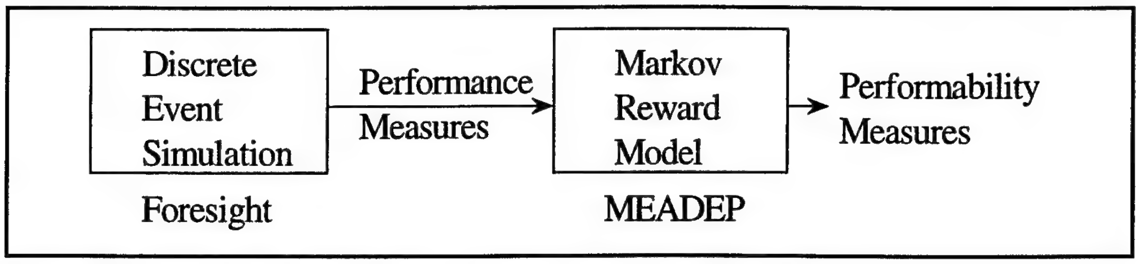

Figure 11 Combined Use of Simulation and Analytical Modeling

3.3.1 Simulation Model

Assume there are five types of input requests:

- Flight Control
- Navigation
- Flight Management
- Defense
- Payload

A request can be routed to any available processor, in the sequence of processor 1, 2, and 3. If no processor is available, the request is dropped. All requests have the same priority and follow the "first come, first service" rule. The performance measure to evaluate is the fraction of the dropped requests for each type of request. The hypothetical simulation parameters for the five types of input are listed in Table 5.

Table 5 Hypothetical Simulation Parameters

Input Type	**Arrival**		**Processing**	
	Arrival Rate (per second)	**Interarrival Time Distribution**	**Mean Processing Time (seconds)**	**Processing Time Distribution**
Flight Control	10	Constant	0.04	Exponential
Navigation	2	Constant	0.2	Exponential
Flight Management	0.01	Exponential	4	Exponential
Defense	0.005	Exponential	8	Exponential
Payload	0.0025	Exponential	16	Exponential

This simulation model was implemented with both Foresight and Designer. The simulated real time was 100 hours. The simulation results are listed in Table 6. First of all, the execution time for Foresight is much longer than that for Designer (450 vs. 30 minutes), although the underlying processor speed for Foresight is faster than that for Designer. The results generated by the two tools are different. This is probably due to the difference between the event scheduling algorithms used in the tools. For example, the "drop count" for the Navigation request in Designer is much higher than that in Foresight (7380 vs. 1264). This implies that the event Navigation is scheduled with a higher priority in Foresight than in Designer.

Table 6 Simulation Results for 3-Processor System

Input Type	Foresight			Designer		
	Total Count	**Drop Count**	**Drop Fraction**	**Total Count**	**Drop Count**	**Drop Fraction**
Flight Control	3600000	42688	0.011858	3600000	31310	0.008698
Navigation	72000	1264	0.001756	72000	7380	0.010250
Flight Management	3514	92	0.026181	3547	91	0.025655
Defense	1812	60	0.033113	1786	55	0.030795
Payload	927	33	0.035599	853	38	0.044549
Execution Time	450 minutes			30 minutes		
Platform	Pentium II 300, Windows NT			Sun SPARCstation 5, Solaris		

The results shown in Table 6 were generated from a 3-processor model. Similar results were also generated from a 2-processor model and a 1-processor model. In the 2-processor model, the third processor is assumed to have failed. In the 1-processor model, the other two processors are assumed to have failed. The results generated by Foresight for the three simulation models will be used in a Markov reward performability model discussed below.

3.3.2 Analytical Model

A Markov reward performability model was defined with MEADEP as shown in Figure 12. In the model, state S_i represents that i processors are functioning properly. The number below the symbol S_i is the reward assigned to state S_i. This number is the fraction of the Flight Control requests that

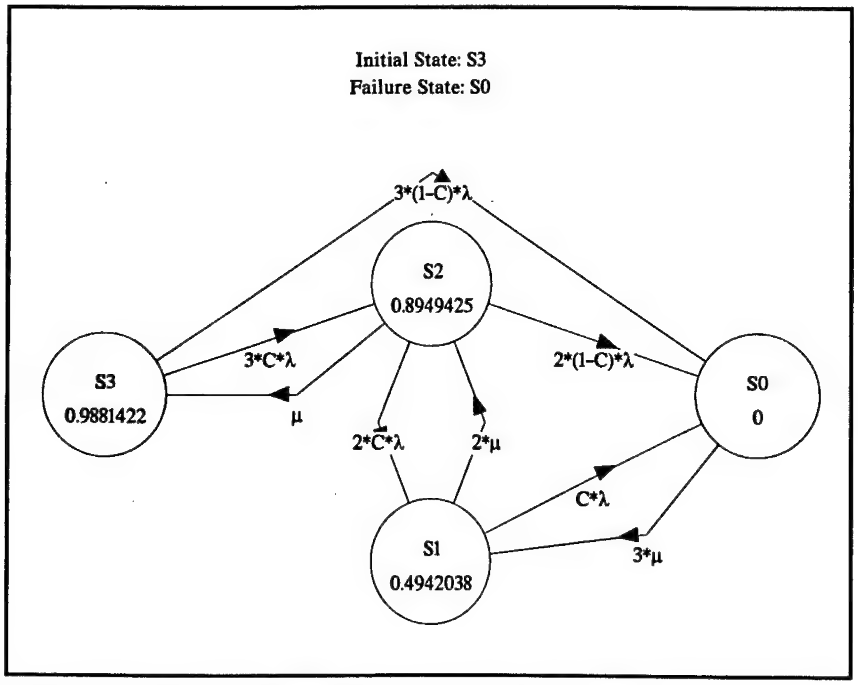

Figure 12 Markov Reward Model for Flight Control

were timely processed in the *i*-processor system (from the performance model) and calculated as 1−Drop Faction (Table 6). For example, in state S_2, the reward is 0.8949425. This number is the fraction of the Flight Control requests that can be timely processed in the 2-processor configuration which is equivalent to state S_2 where one processor has failed. The expected reward, or performability, evaluated from this model is the probability that the Flight Control requests can be timely processed under fault conditions. This performability model can be similarly applied to other types of input (Navigation, etc.).

Assume the processor failure rate is 10^{-3}/hour, the repair rate is 0 (i.e., a non-repairable system), and the coverage is 0.99 (the reconfiguration is 99% successful after a processor fails). The performability over a time range from 0 to 100 hours generated by MEADEP for the 3-processor system is drawn (the lower curve) in Figure 13. The curve is similar to a reliability curve, but here it represents performance related reliability. If the processor failure rate is improved to 10^{-4}/hour, this performability measure would be increased from 0.95 to 0.985 for a 100 hour mission, as shown by the upper curve in the figure. This evaluation could be used to select the quality level of COTS components to meet the performability requirements.

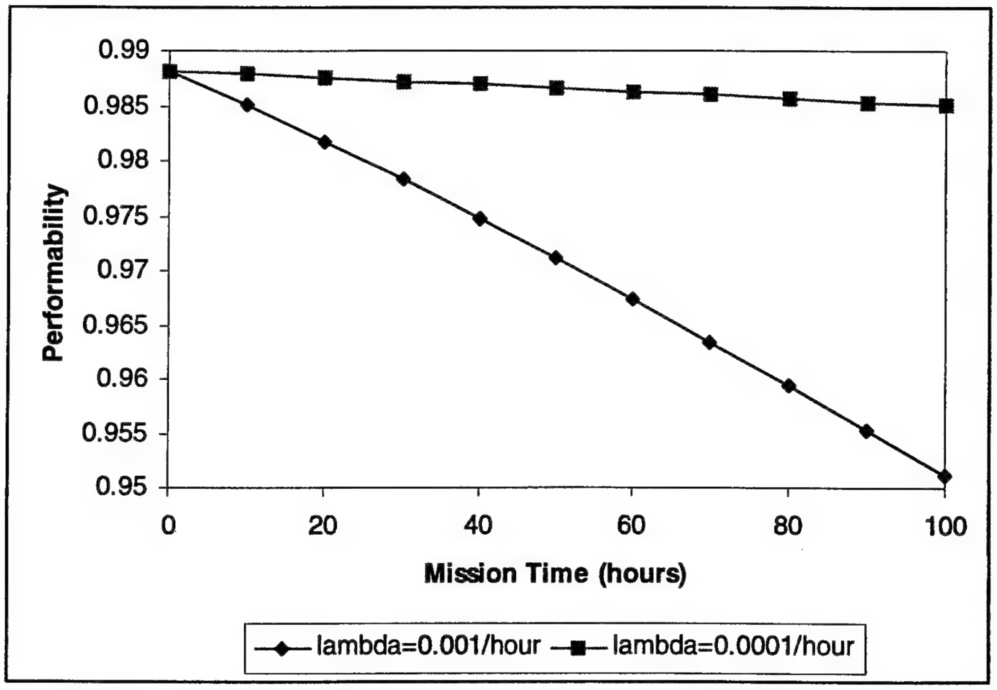

Figure 13 Probability of Not Missing "Flight Control" Requests

3.3.3 Fault Injection Approach

Although the above hybrid simulation/analytical model is good for many situations, sometimes detailed fault behaviors (e.g., fault latency due to the time difference between fault arrival and fault activation) and recovery processes (e.g., rollback recovery in software modeling) are difficult to model analytically and the simulation approach has to be used. A reasonable question is: can the above performability be evaluated by the simulation model only? The answer is yes, but it is costly. Now we use the UAV model to show why this approach is costly.

In order to evaluate performance under fault conditions, fault injection has to be used in the simulation. Because this is system level modeling, system level fault models [Iyer96] should be used. For demonstration purposes, the fault model applied here is simple: transient faults arrive at a constant rate with the exponential distribution. Each fault causes a processor failure and triggers a restart of the processor. During this restart (recovery) process, the processor cannot process any input requests. When the recovery is complete, the processor is restored to the normal state. We assume faults on different processors are independent and the reconfiguration after a processor failure is always successful (perfect coverage). Thus we add three independent fault injectors to the UAV 3-processor model with each processor having a fault injector.

The typical fault rate range is 10^{-3}/hour or lower. This contrasts with the range of performance parameter rates which are typically in the order of fractional seconds. In the simulation that handles both performance parameters and fault injection, a fault has to wait for hundreds of thousands of time units to occur, because the time unit is determined by the highest performance parameter occurrence rate. The execution time of fault injection simulation can thus be prohibitively long. Considering the UAV example, assume we want to observe at least 10 faults with an arrival rate of 10^{-3}/hour. Then we need to simulate at least 10,000 hours of real time. Table 6 shows that Foresight takes 450 minutes (7.5 hours) to simulate 100 hours of real time. This translates the 10,000 hours of simulated time to 750 hours, or over a month of execution time. If the failure rate is improved to 10^{-4}/hour, the execution time would require 10 months.

The cost of this approach has exceeded what this research project can afford. In order to demonstrate the fault injection approach, we have to increase the failure rate in our simulation experiments. In the first simulation, the failure rate was set to 0.1/hour, or 10 hours of Mean Time To Failure (MTTF). In the second simulation, MTTF was doubled (20 hours). In the third simulation, MTTF was doubled again (40 hours). This process was repeated until the a limitation in time is reached. The processor recovery time was assumed to be 10 minutes and the coverage was assumed to be 1 (perfect coverage). Foresight was used in this study. A software crash was encountered whenever the simulated time reached 245 hours.[3] Thus the simulated time had to be set to a value less than 245 hours. That is, the time limitation mentioned above has to be less than 245 hours. In this study, 200 hours of real time was simulated in each experiment, which required 15 hours of execution time.

Results for the "Flight Control" request generated by Foresight are listed in Table 7. The corresponding results generated by MEADEP are also shown in the table for comparisons. The longest MTTF for which Foresight can generate results is 160 hours, because the simulated time is 200 hours (an MTTF longer than 200 hours may not generate even a single failure event in the simulation). It took a total of 75 hours of execution time in generating the five Foresight results listed in the table. In the "# of Faults" row, there are three numbers which represent the faults injected to the three processors. As the number of faults injected decreases, the relative error of the result (compared to the MEADEP result) increases. To reduce this error, longer time needs to be simulated. In this particular example, 20 faults need to be injected in order to obtain a result with the relative error less than 1% (Column 2). It is thus infeasible to obtain accurate results for a highly reliable (low failure rate) system by using this simulation approach.

[3]Nu Thena has repeated this problem and is diagnosing it.

Table 7. Drop Fraction for "Flight Control" under Failure Conditions

MTTF	10 hours	20 hours	40 hours	80 hours	160 hours	1000 hours
MEADEP	0.016688	0.014231	0.013034	0.012443	0.012150	0.011904
Foresight	0.016717	0.014668	0.013500	0.012954	0.012707	0.012282
# of Faults	21, 18, 20	13, 11, 13	7, 6, 5	4, 2, 3	1, 0, 2	interpolation
Error	0.17%	3.07%	3.57%	4.11%	4.58%	3.18%

Because of infeasibility of simulating low failure rate systems with the regular fault injection approach, other methods have to be used to reduce the execution time. One approximate method is interpolation. In the above example, based on the result for the MTTF of 160 hours and the result from the simulation without fault injection (MTTF is infinite), it can be inferred that the result for the MTTF of 1,000 hours must be a point between 0.012707 and 0.011858. This point can be interpolated in this range. The "1000 hours" column in the table shows the mid value. This estimate is better than that obtained by the linear interpolation. The non-linearity property of the curve shown in Figure 14 indicates that the error of a linear interpolation can be large.

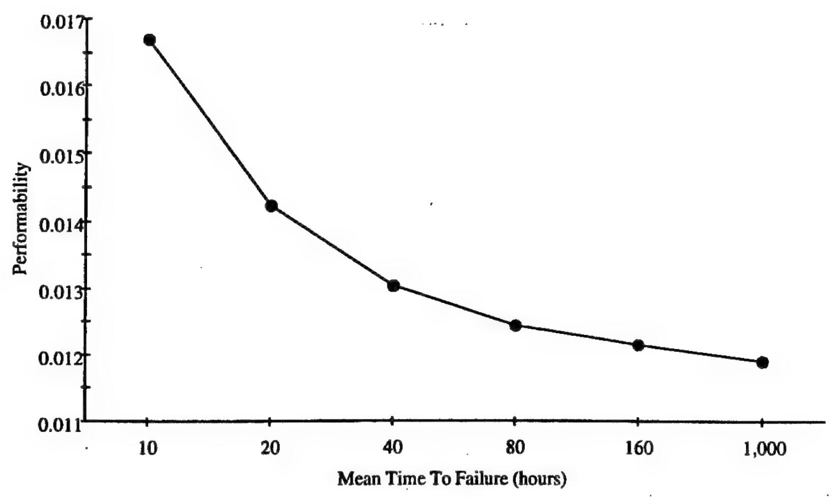

Figure 14 Results Generated by MEADEP Parametric Analysis

3.3.4 Importance Sampling

Now we examine a technique that can accelerate fault injection simulations: *importance sampling*. Importance sampling is a statistical method to reduce sampling size while keeping estimates obtained from the sample unbiased at a high level of confidence. The method has been used to evaluate dependability measures from rare events in Monte Carlo simulations to solve large Markovian models [Conway87]. The idea is to increase the failure rate and then to scale the results back using a mathematical function called likelihood ratio. Although it can be difficult to determine the likelihood ratio function, for the above example, the likelihood ratio can be easily determined.

We define the *performance loss* as the difference between the performance measure obtained from the simulation with fault injections and that obtained from the simulation without fault injections and denote it by θ. In the above example, it is the fraction of dropped input requests caused by failures. For the fault injection simulation with MTTF of 10 hours, θ is calculated by subtracting the result of the simulation without fault injection (0.011858) from the simulation result in the "10 hours" column (0.016717) in Table 7. The resultant number is 0.004859.

The unavailable time of system resources caused by failures is proportional to the failure rate. Thus, for performance measures that depend on the availability of system resources, the likelihood ratio, Λ, is just

$$\Lambda = \frac{MTTR}{MTTR'} \tag{1}$$

where *MTTR* is actually used in the simulation and *MTTR'* is the parameter we want to, but cannot use in the simulation because of the high cost. According to the importance sampling theory, the performance loss to evaluate for the system with failure rate 1/*MTTR'*, θ', can be calculated by

$$\theta' = \theta \times \Lambda \tag{2}$$

For example, for the *MTTF'* of 1000 hours, $\Lambda = 0.01$ and the corresponding $\theta' = 0.004859 \times 0.01 = 0.00004859$. The performability measures for the MTTF of 20 to 1000 hours shown in Table 7 can then be calculated from θ'. Since we only simulated a system with the MTTF of 10 hours and derive other measures from the result of this simulation using the importance sampling method, the execution time is reduced by a factor of 100.

Table 8 lists results calculated using Equations (1) and (2) and compares these results with the MEADEP results. Comparing Table 8 with Table 7, we see the errors of the importance sampling results are much less than those from simulations with a duration less than enough time (number of generated faults is less than 20) and from the interpolation. Table 8 shows better results because: (1) importance sampling is a scientific method, and (2) the simulation result upon which the importance sampling method was based (the "10 hours" column) is a good estimate (with a very small error).

Table 8. Drop Fraction for "Flight Control" Derived by Importance Sampling

MTTF	10 hours	20 hours	40 hours	80 hours	160 hours	1000 hours
MEADEP	0.016688	0.014231	0.013034	0.012443	0.012150	0.011904
Importance Sampling	0.016717	0.014287	0.013073	0.012465	0.012161	0.011907
Error	0.17%	0.4%	0.3%	0.18%	0.09%	0.02%

3.3.5 Summary of Case Study 2

Table 9 summarizes this case study. It is seen that the combined use of performance simulation and analytical dependability modeling is a more cost effective approach than the performance/fault injection simulation approach. Whenever possible, this approach should be used. If the performance/fault injection simulation approach has to be used to model detailed fault behaviors and recovery processes, to make this approach feasible for evaluating highly reliable systems, the following methods can be used:

- Approximate the target measure by the interpolation method.
- Incorporate importance sampling in the simulation.
- Enhance the speed of the simulation tool.

We recommend that the second and third methods be used whenever possible.

Table 9. Summation of Case Study 2

Approach	Simulated Time	Comments
Simulation/analytical modeling	300 hours	100 hours for each additional system state Fast parametric analysis (e.g., on repair time)
Conventional simulation	20,000 hours	Infeasible for low failure rate
Simulation/interpolation	1,100 hours	Error of the results may be large
Importance sampling simulation	300 hours	In parametric analysis, simulation needs to repeat for each set of new parameters

4. High Level Tool Design

Based upon the above tool evaluation results, we propose a tool set that supports:

- combination of performance simulation and analytical dependability modeling,
- fast simulation that incorporates both performance modeling and fault injection.

The tool set is based upon Foresight and MEADEP. The reason for selecting these tools is because they both have high scores in terms of the evaluation metrics we developed. Another consideration is that they both run on Windows NT which provides a good user interface and supports many other tools (e.g., word processors) that can easily interface with the selected tools. However, in order to make these tools meet our needs, they should be significantly improved. In the following subsections, we identify the new functionality and improvements that ought to be made on these tools.

4.1 Improvements to Foresight

Reliability The software crash in running the UAV model has been repeated by the Foresight vendor, Nu Thena. A reliable simulation tool should be able to run an application for a long time without crash. This problem and any other reliability problems encountered by users should be resolved by the vendor.

Speed The running speed of Foresight is slow. This is probably due to the low performance of the simulation engine in the interactive simulation environment. However, the Foresight vendor provides an additional tool called CoderC which translates Foresight executable models into C code that can then be compiled and linked with a run-time library to create a stand-alone executable model. According to the vendor, the code generated models may run 15 to 30 times faster (execution speed is dependent upon model) than the graphical models in the Foresight's interactive simulation environment. However, our experiments showed that CoderC did not work for the UAV model on the Windows NT platform and the model was sent to the Nu Thena technical support for diagnosis. To offer fast simulation, Nu Thena should make this tool work for applications on Windows NT.

New Statistical Functions As we stated in Section 2, in addition to the exponential and normal function, dependability modeling also needs the Weibull and lognormal functions. These functions should be added into the standard library of Foresight.

Fault Injection Support Elements that support fault injection should be added into the Foresight library. At least two types of library elements are needed: fault generator and fault injectable resource. A fault generator periodically generates faults according to a statistical distribution (exponential, Weibull, etc.). A fault injectable resource is a library resource element that can accept fault injection. All existing library resource elements need to be examined to determine if they support fault injection. For example, the library element Process Resource can be manipulated by the mini-spec call reduceProRes (name, amount, preempt). This function could be used to inject faults. However, the similar function does not exist for the library element Basic Resource. A new function that supports fault injection should be introduced for this resource.

Primitive Fault Tolerant Architecture models Some primitive fault tolerant architecture models, such as primary/standby modular redundancy and triple modular redundancy models, should be developed and included in the tool package. These models should provide parametric functions for users to invoke.

Documentation Support Although the Foresight NT version has the ability to export a graphical model diagram to the postscript format, the generated postscript file cannot be recognized by other tools such Ghostview and Microsoft Word. This problem should be resolved by the vendor. If possible, the Windows Meta File (WMF) format should also be supported in the Foresight NT version, because this format is more widely used than the postscript format in Microsoft Windows applications.

More User-Friendly On-Line Help The current on-line help system in Foresight provides limited information (no detailed definitions for library elements and associated functions) and lacks user-friendly features such as index and search capabilities. The on line help system should be improved by adding these capabilities and expanding its contents.

Importance Sampling Support An off-line interactive analysis tool is desired for evaluating final results based on the results obtained from a simulation without fault injection and a simulation with importance sampling fault injection. This analysis tool should be able to interface with the output of the simulation tool and provide a graphical user interface.

4.2 Improvements to MEADEP

MEADEP has a graphical user interface which allows the user to draw models on the screen. This has been a step forward from the text specification of models used in SHARPE. However, the user still has to map the system architecture to a dependability model which is not always an easy task. This is specially true for developing Markov models. It would be very useful to have an I*ntelligent Model Generator* which helps the user who understands the system architecture but does not have the expert knowledge in dependability modeling to generate a graphical dependability model based on queries and answers.

To support the simulation/analytical modeling approach, the intelligent model generator is also used to generate a high level resource-consumer type simulation model for performance evaluation. A corresponding simulation engine needs to be developed. The interconnection between the simulation model and the dependability model can be specified by the user (e.g., by reward assignment). Although the simulation models supported by the simulation engine are only a subset of those supported by Foresight, these models are likely to satisfy most applications in the early life cycle phases. Since the performance simulation model, the analytical dependability model, and connections between the two models are all generated automatically, the intelligent model generator provides a high degree of modeling automation environment.

The current MEADEP version has two modeling related modules: (1) Model Generator which is used to draw graphical models and to generate text modeling files, and (2) Model Evaluator which takes the text modeling file as input and generates results. Figure 14 shows these two modules and the new modules to be added (in the dashed frame): *Model Wizard* which is the intelligent model generator and *Simulation Engine*. The ellipses represent files. The relationships between program modules and files are expressed by lines and arrows connecting them. A line with a one direction arrow means the connected file can be read or written by the connected program module only. A line with two arrows in both directions means the connected file can be read and written (i.e., edited) by the connected program module.

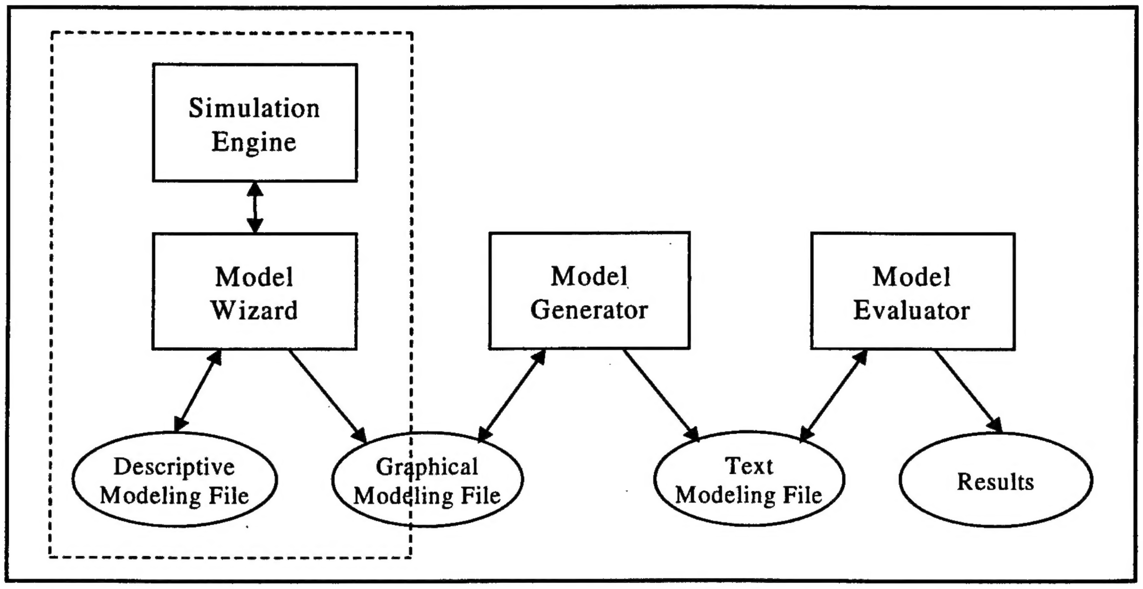

Figure 15 New Modules Added to MEADEP

Model Wizard will be a dialog-driven program. In each dialog, the user is given multiple choices to select by clicking mouse buttons. For example, an initial dialog box may show the following questions:

- How many components will be in this diagram? (Type a number)
- Are these components all required for the system to function? (Yes/No)
- Can a failure of one component cause a failure of another component (e.g., due to an unsuccessful reconfiguration)? (Yes/No)

Different answers may invoke a different subsequent dialog box. For instance, if the answers to the above questions are "3, No, Yes," a Markov model should be used for the top level diagram and the next dialog may ask the following questions:

- How many components are required for the system to function (1 or 2)?
- Do these components have the same failure rate and recovery rate parameters (Yes/No)?

......

When this "questions and answers" procedure is completed, a high-level model specification is constructed. The specification has a tree structure and can be edited by the user. Model Wizard can also generate a simulation model based on similar queries. The simulation results can be designated as rewards for generated Markov models. The high-level model specification and associated parameters can be saved in the *descriptive modeling file* for reuse. Based on this descriptive modeling file, a graphical modeling file can be generated by Model Wizard. When the user opens this graphical modeling file with Model Generator, the generated model will be graphically displayed on the screen. The graphical modeling file can then be translated by Model Generator to a text modeling file which, in turn, can be evaluated by Model Evaluator to generate final results.

Other Improvements to MEADEP

Full Support for Performability Evaluation MEADEP was initially designed for dependability evaluation. The state reward assignment is restricted to the range between 0 and 1. The results generated are all dependability measures. Although the "Availability" result may be performance related by assign a system capacity parameter to the state reward, it is still restricted to the range between 0 and 1. Performability measures, such as the number of "Flight Control" requests dropped per second in Case study 2, may exceed 1. It is necessary to remove these restrictions and add measures in the performability category to the output of MEADEP.

Extend the Size Limit on the Modeling File The maximum size of the text modeling file that can be handled by Model Evaluator is currently 64K bytes. This is due to the limitation of the Microsoft Foundation Class CEditView which was used to implement Model Evaluator. For the same reason, similar limitations can also been seen on the text editor "Notepad" that comes with Windows 95/98/NT. Although most MEADEP applications do not exceed this size limit, it should be extended to a higher level to meet possible higher demands in the future.

More User Friendly On-Line Help The current on-line help system in MEADEP has only a limited search capability (keywords) and lacks the index capability. The on-line help system should be improved by adding these capabilities and expanding its contents.

5. Conclusion

This research has overcome the problem posed by widely differing time scales used for performance and dependability measures and has shown the feasibility of combining performance and dependability modeling in coordinated software tools, a capability that is needed but had not been adequately met. The combined modeling capability permits top level decision makers to evaluate system designs in cases where a component or software failure as well as impaired performance can jeopardize the ability to meet mission objectives. Actions taken to improve dependability can detract from performance (e. g., the message passing in voting structures), and measures taken to improve performance can detract from dependability (e. g., speeding up transmission rates will increase errors due to noise). These interactions make a combined model very valuable for both military and commercial applications. We have shown the feasibility of a combined model.

In the first task of this Phase I effort we developed a set of evaluation metrics and applied them to several representative performance/dependability modeling tools. Experimental evaluation was also performed on these tools by case studies. The evaluation has identified two cost effective approaches to modeling performability (performance + dependability) for fault tolerant avionic systems:

- combination of discrete event performance simulation and analytical dependability modeling,
- integrated discrete event simulation for performance and dependability evaluation, with reasonable execution time made possible by importance sampling; this permits fault injection for extremely flexible and high fidelity modeling of performance under exception conditions.

For the first approach, we showed how to relate performance measures (system capacity) to system states in terms of component failures. For the second approach, we enabled importance sampling to be used by introducing the performance loss measure to compute the likelihood ratio which is needed to calculate the final (true) measure based on results obtained from a simulation with highly magnified fault rates.

To automate these approaches, we propose a tool set to be developed in Phase II that integrates a commercial performance simulation tool and a commercial dependability modeling tool. The Phase II research will also extend the theoretical basis for importance sampling in this modeling context. Necessary improvements/extensions to the existing tools and additional auxiliary tools to support the above approaches have been identified. The synthesized tool set is not only good for modeling fault tolerant avionic systems, but also has a great commercialization potential.

The research reported on here also advances the state-of-the-art in software tools for modeling fault tolerant (FT) systems, thereby promoting improved fault tolerance effectiveness and lower life cycle cost. This capability, too, is applicable to military as well as commercial uses.

References

[Cadence98] Cadence Design Systems, Inc., *BONeS DESIGNER User's Guide*, March 1998.

[Conway87] A. E. Conway and A. Goyal, "Monte Carlo Simulation of Computer System Availability/Reliability Models," *Proceedings of the 17th International Symposium on Fault-Tolerant Computing*, June 1987, pp. 230-235.

[Goswami97] K. K. Goswami, R. K. Iyer and L. Young, "DEPEND: A Simulation-Based Environment for System Level Dependability Analysis," *IEEE Transactions on Computers*, Vol. 46, No. 1, Jan. 1997, pp. 60-74.

[Iyer96] R. K. Iyer and D. Tang, "Experimental Analysis of Computer System Dependability," *Fault-Tolerant Computer System Design*, D. K. Pradhan (Ed.), Prentice Hall PTR, Upper Saddle River, NJ, 1996, pp. 282-392.

[Laprie85] J. C. Laprie, "Dependable Computing and Fault Tolerance: Concepts and Terminology," *Proceedings of the 15th International Symposium on Fault-Tolerant Computing*, June 1985, pp. 2-11.

[Laprie95] J. C. Laprie, "Dependable Computing: Concepts, Limits, Challenges," *Special Issue of the 25th International Symposium on Fault-Tolerant Computing*, June 1995, pp. 42-54.

[Lee95] I. Lee and R. K. Iyer, "Software Dependability in the Tandem GUARDIAN System," *IEEE Transactions on Software Engineering*, Vol. 21, No. 5, May 1995, pp. 455-467.

[Meyer80] J. F. Meyer, "On Evaluating the performability of Dependable Computing Systems," *IEEE Transactions on Computers*, Vol. 29, No. 8, Aug. 1980, pp. 720-731.

[NuThena96] Nu Thena Systems, *Foresight User's Guide*, Release 4.10, September 1996.

[Sahner87] R. A. Sahner and K. S. Trivedi, "Reliability Modeling Using SHARPE", *IEEE Transactions on Reliability,* vol 36, February 1987, pp. 186-193.

[Sahner96] R. A. Sahner, K. S. Trivedi and A. Puliafito, *Performance and Reliability Analysis of Computer Systems: An Experimental-Based Approach Using the SHARPE Software Package*, Kluwer Academic Publishers, 1996.

[Sanders95] W. H. Sanders, W. D. Obal II, M. A. Qureshi, and F. K. Widjanarko, "The UltraSAN Modeling Environment," *Performance Evaluation*, Vol. 24, No. 1, Oct/Nov 1995, pp. 89-115.

[Tang98] D. Tang, M. Hecht, J. Handal and L. Czekalski, "MEADEP and Its Applications in Evaluating Dependability for Air Traffic Control Systems," *Proceedings of the 1998 Annual Reliability and Maintainability Symposium*, Jan. 1998, pp. 195-201.

[Tang99] D. Tang, M. Hecht, A. Rosin and J. Handal, "Experience in Using MEADEP," To appear in *Proceedings of the 1999 Annual Reliability and Maintainability Symposium*, Washington DC, Jan. 18-21, 1999.